Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 4 (2011) 1535–1544

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

145 TFlops Performance on 3990 GPUs of TSUBAME 2.0 Supercomputer for an Operational Weather Prediction

Takashi Shimokawabe[a,1,*], Takayuki Aoki[b,d], Junichi Ishida[c], Kohei Kawano[c], Chiashi Muroi[c]

[a] *Department of Energy Sciences, Tokyo Institute of Technology*
[b] *Global Scientific Information and Computing Center, Tokyo Institute of Technology*
[c] *Numerical Prediction Division, Japan Meteorological Agency*
[d] *Japan Science and Technology Agency, CREST*

Abstract

Numerical weather prediction is one of the major applications in high performance computing and demands fast and high-precision simulation over fine-grained grids. While utilizing hundreds of CPUs is certainly the most common way to get high performance for large scale simulations, we have another solution to use GPUs as massively parallel computing platform. In order to drastically shorten the runtime of a weather prediction code, we rewrite its huge entire code for GPU computing from scratch in CUDA. The code ASUCA is a high resolution meso-scale atmosphere model that is being developed by the Japan Meteorological Agency for the purpose of the next-generation weather forecasting service. The TSUBAME 2.0 supercomputer, which is equipped with 4224 NVIDIA Tesla M2050 GPUs, has started operating in November 2010 at the Tokyo Institute of Technology. A benchmark on the 3990 GPUs on TSUBAME 2.0 achieves extremely high performance of 145 TFlops in single precision for $14368 \times 14284 \times 48$ mesh. This paper also describes the multi-GPU optimizations introduced into the ASUCA porting on TSUBAME 2.0.

Keywords: Numerical weather prediction, High performance computing, GPGPU

1. Introduction

Weather forecasting is an indispensable parts in our daily lives and business activities, needless to say, for natural disaster preventions. The atmosphere has a very thin thickness to compare with the Earth diameter. In the previous atmosphere codes, the force balance between the gravity and the pressure gradient was assumed in the vertical direction and we call them hydrostatic models. Recently it is widely recognized that the vertical dynamics of the water vapor should be taken into consideration in cloud formations. Three-dimensional non-hydrostatic model describing up-and-down movement of the air have been developed in the weather research. It is highly demanded to forecast detailed weathers such as unexpected local heavy rain, and the high resolution non-hydrostatic models is desired to be carried out with fine-grained grids.

* *Email address:* shimokawabe@sim.gsic.titech.ac.jp (Takashi Shimokawabe)
[1] Corresponding author

Selection and/or peer-review under responsibility of Prof. Mitsuhisa Sato and Prof. Satoshi Matsuoka
doi:10.1016/j.procs.2011.04.166

Numerical weather prediction is one of the major applications in the field of high performance computing (HPC). We have heavy computational load to run the high-resolution weather models. A next-generation atmosphere simulation model WRF (Weather Research and Forecasting) [1] is a world standard code developed at the NCAR (National Center for Atmospheric Research), UCAR(University Corporation for Atmospheric Research) and so on in the United States and supported by worldwide researchers. The WRF has been scored 50 TFlops on one of the world top-class supercomputers [2].

Recently, exploiting Graphics Processing Units (GPUs) for general-purpose computation, i.e. GPGPU, has emerged as a high-performance computing device to accelerate many applications and become an active research area in parallel computing. Although GPUs have been developed for the rendering purpose of computer graphics, after CUDA [3] was released by NVIDIA as the GPGPU-programming framework in 2006, it allows us to use the GPU easily as GPGPU device. It is well known that GPUs successfully accelerate scientific computing, such as Computational Fluid Dynamics (CFD) [4, 5], Fast Fourier Transform [6], and astrophysical N-body simulations [7], dozens time faster than a conventional CPU, thanks to their high performance of floating point calculation and wide memory bandwidth with relatively low cost.

In the numerical weather prediction, it was reported that a computationally expensive physics module of the WRF model was accelerated by using a GPU [8, 9]. The acceleration for the WRF Single Moment 5-tracer (WSM5) microphysics itself achieved a twenty-fold speedup [8]. These efforts, however, only result in a minor improvement (e.g., 1.3× in [8]) in the overall application time due to the partial GPU porting of the entire code. In addition, since the other subroutines in the code still run on the CPU and all the variables were allocated on the main memory, it is necessary to transfer the data between device and host memories through the PCI Express bus every call of the GPU kernel-function and it leads to degrade in performance.

Numerical weather models consist of a dynamical core and physical processes. While the dynamical core computes time integration of forecast variables, such as winds, atmospheric pressure and humidity, by solving fluid dynamics equations, the physical processes strongly depend on parametrizations related to such microphysics as condensation of water vapor, cloud physics, and rain, which are composed of small and relatively independent modules and able to be easily changed to other modules. The previous works adopted the acceleration of the computationally intensive physics modules by the GPU. However, to fully exploit the benefits of GPUs, the entire part should be executed on GPUs with minimizing the communication between host and device.

We are currently working on full GPU implementation for ASUCA [10] – a next-generation high resolution mesoscale atmospheric model being developed by the Japan Meteorological Agency. We have successfully implemented its dynamical core and a portion of physics processes as a full GPU application, representing an important step toward establishing a complete framework for the full GPU-based ASUCA. In our previous paper [11], we have presented that ASUCA have achieved 15.0 TFlops using 528 GPUs on the TSUBAME 1.2 Supercomputer [12] at the Tokyo Institute of Technology. As the successor to TSUBAME 1.2, the TSUBAME 2.0 Supercomputer, which is equipped with 4224 NVIDIA Tesla "Fermi" M2050 GPUs, has started operating in November 2010 and has become the first petascale supercomputer in Japan. In this paper, we show the optimization developed in [11] is of benefit to the computation on TSUBAME 2.0 and demonstrate the performance of both single- and multi-GPU computation of ASUCA using the NVIDIA Tesla M2050 GPUs of TSUBAME 2.0. As a result, our multi-GPU version that combines distributed GPUs over InfiniBand-connected nodes with MPI achieves very high performance of 145.0 TFlops in single precision with 3990 GPUs on TSUBAME 2.0. We also focus on the difference in the effects of the optimization on between TSUBAME 2.0 and TSUBAME 1.2.

2. Weather Simulation code ASUCA

In this section, we review ASUCA (Asuca is a System based on a Unified Concept for Atmosphere), a next-generation high resolution mesoscale atmospheric model being developed by the Japan Meteorological Agency (JMA) [10]. The ASUCA succeeds the Japan Meteorological Agency Non-Hydrostatic Model (JMA-NHM) [13] as an operational non-hydrostatic regional model. We do not show the equations used in ASUCA due to the page limitation, and these detail are described in [10, 11].

First, we have implemented the dynamical core as a first step to develop the full GPU version of the ASUCA. In the ASUCA, a generalized coordinate and flux-formed non-hydrostatic balanced equations are used for the dynamical

core. The time integration is carried out by a fractional step method with the horizontally explicit and vertically implicit (HE-VI) scheme using a time-splitting method [14]. One time step consists of short sub-steps and a long time step. The horizontal propagation of sound waves and the gravity waves with implicit treatment for the vertical propagation are computed in the short time step with the second-order Runge-Kutta scheme. The long time step is used for the advection of the momentum, the density, the potential temperature and the water substances, the Coriolis force, the diffusion and other effects by physical processes with the third-order Runge-Kutta method. The above matters are almost same as those employed in the WRF model. In the present ASUCA, the physical core is still being developed and a Kessler-type warm-rain model, which is used in the JMA-NHM [13], has been implemented for the cloud-microphysics parameterization describing the water vapor, cloud water, and rain drops.

3. NVIDIA GPUs and The TSUBAME 2.0 Supercomputer

This section describes the TSUBAME 2.0 supercomputer in the Tokyo Institute of Technology, which is used for our evaluation. The NVIDIA Tesla "Fermi" M2050, which is introduced into TSUBAME 2.0, has 3 GB GDDR5 SDRAM device memory and 448 CUDA Core. The peak performance of a GPU is 1030 GFlops and 515 GFlops in single- and double- precision, respectively. There are 32 CUDA cores in a streaming multiprocessor (SM) as a SIMD (single instruction, multiple data stream) unit; thus each GPU contains 14 SMs. The on-board device memory (also called *global memory* in CUDA), shared by all the SMs in the GPU, provides 148 GB/s peak bandwidth in a Tesla M2050. In spite of its excellent memory bandwidth, each access to the device memory takes 400 to 600 clock cycles. To hide this overhead and harness locality, Fermi has configurable 64 kB private first-level caches in each SM and a 768 kB shared second-level cache. The first-level cache can be partitioned as 16 kB/48 kB or 48 kB/16 kB; one partition is shared memory used as extremely fast scratch-pad memory that stores temporary data accessible by all 32 cores in the SM, the other partition is an L1 cache.

The TSUBAME 2.0 supercomputer in Tokyo Institute of Technology is equipped with 4224 NVIDIA Tesla M2050 GPUs. Each node of TSUBAME 2.0 has three Tesla M2050 attached to the PCI Express bus 2.0 ×16 (8 GB/s), two QDR InfiniBand and two sockets of the Intel CPU Xeon X5670 (Westmere-EP) 2.93 GHz 6-core. All the nodes are connected to the fat-tree interconnection with 200 Tbps bi-section bandwidth. In order to achieve multi-GPU computing on TSUBAME 2.0, we use two types of communication libraries: (1) a MPI library for inter-node communication, and (2) the CUDA runtime library for communication between CPUs and GPUs. In our evaluation, the Open MPI library version 1.3.3 and CUDA version 3.1 are used.

4. Multi-GPU Computing of ASUCA

In our previous paper [11], we presented multi-GPU implementation of ASUCA and its performance for NVIDIA Tesla S1070 GPUs on TSUBAME 1.2. It has been confirmed that several optimizations are also effective for multi-GPU computing on TSUBAME 2.0. In this section, we review those optimization techniques.

4.1. Basic Approach

The original ASUCA is being developed in the Fortran language at the JMA. We rewrote it to the GPU code from scratch in CUDA. Before implementing the ASUCA on GPU, we changed the Fortran ASUCA code to C/C++ language in order to change the element order of the 3-dimensional array to improve the memory access performance for the GPU computing. Using the C/C++ code as reference, we developed the CUDA code of ASUCA. All computational modules are executed on a GPU accessing to the variables allocated on its video memory. In the beginning of the execution, the host CPU reads the initial data from the input files onto the host memory, and transfers them to the video memory on the GPU board. The GPU carries out all the computational modules inside the short and long time-step loops. When the forecast data are completed on the GPU, the minimal data are transferred to the host CPU memory to reduce the communication between CPU and GPU.

4.2. Single GPU Optimizations

Our final destination is to develop the multi-GPU version of ASUCA. First, We describe the single GPU implementation of ASUCA.

All the computations in a dynamical core, such as the 3-dimensional advection computation, are strongly memory bound and it is very effective to reduce the access to the global memory. In order to archive high performance, the shared memory and the registers are used as software managed cache in CUDA kernels. In order to exploit data stored in shared memory efficiently in parallel computation, we calculate all the elements of the computational domain as follows; the kernel function for the advection, for example, is invoked with $(nx/64, nz/4, 1)$ blocks with $(64, 4, 1)$ threads for a given grid size (nx, ny, nz) of the computational domain. Each thread specifies a point (x, z) and calculate the advection equation on the grid point from $j = 0$ to $j = ny - 1$ marching in the y direction. The advection computation has a four-point stencil in each direction. Since each element in the xz-slice is accessed by several neighbor threads, the data loaded by a thread is stored in the shared memory to be reused by the neighbor threads. On the other hand, neighbor elements aligned in the y direction is used only by a single thread. Thus each thread keeps three y elements in the registers. When we compute the $j + 1$-th plane, the data accessed by several threads have been already loaded by each thread from the global memory into the registers in previous j-th plane computation. In our implementation, we copy these data stored in the registers to the shared memory and reuse them to reduce the global memory access.

4.3. Multi-GPU Optimizations

For large-size problems beyond the video memory on a single GPU, it is necessary to use multiple GPUs. Tesla M2050 has 3 GB of video memory, which can only hold up to a grid of size $256 \times 208 \times 48$ in single precision.

We decompose the whole computational domain in both the x- and y-directions (2-D decomposition) and allocate each sub domain to a GPU since the mesh size is relatively small in the z-direction. Similar to conventional multi-CPU calculations with MPI, multi-GPU computing requires boundary data exchanges between sub domains. Because a GPU cannot directly access to the global memory of the other GPUs, host CPUs are used to bridge GPUs for the data exchange between the neighbor GPUs. This process is composed of the following three steps: (1) the data transfer from GPU to CPU using the CUDA runtime library, (2) the data exchange between nodes with the MPI library, and (3) the data transfer back from CPU to GPU with the CUDA runtime library.

In the case of large-scale problems, the data communication time between GPUs is not ignored in the total execution time. The overlapping technique with the computation is available to hide the communication costs and achieves better performance. The following three overlapping techniques are introduced to ASUCA: (a) Inter-variable overlapping, (b) Kernel division overlapping and (c) Fused kernel overlapping.

(a) Inter-variable overlapping This overlapping exploits inter-variable Independence. When one variable in weather model can be computed independently with another, the boundary exchange for this variable can be overlapping with the computation of another variable. Since the advection of water substances can be computed independently in ASUCA, we apply this overlapping to these computations.

(b) Kernel division overlapping This optimization exploits data independency within a single variable. Since each element of a variable can be computed independently for one calculation, computations for the boundary regions can be performed separately from other calculations for the rest of the domain. By dividing a single kernel into three kernels for the x boundaries, the y boundaries and the inside domain, we can overlap communication for the boundary data exchange with the computation for the inside domain as shown in Figure 1. Note that since the performance of the GPU is often improved for large numbers of threads, dividing kernels to smaller domains degrades the kernel performance itself. However, this overlapping by three divided kernels have a potential to hide communication and result in improving the overall performance.

(c) Fused kernel overlapping Similar to the first optimization, this technique exploits independency between different variables. It attempts to logically fuse multiple kernels for different variables into one so that computation times can be shared by the multiple kernels. Ideally, this would allow more communication time to be hidden by computation times, especially when one kernel spends much longer time in computation than communication.

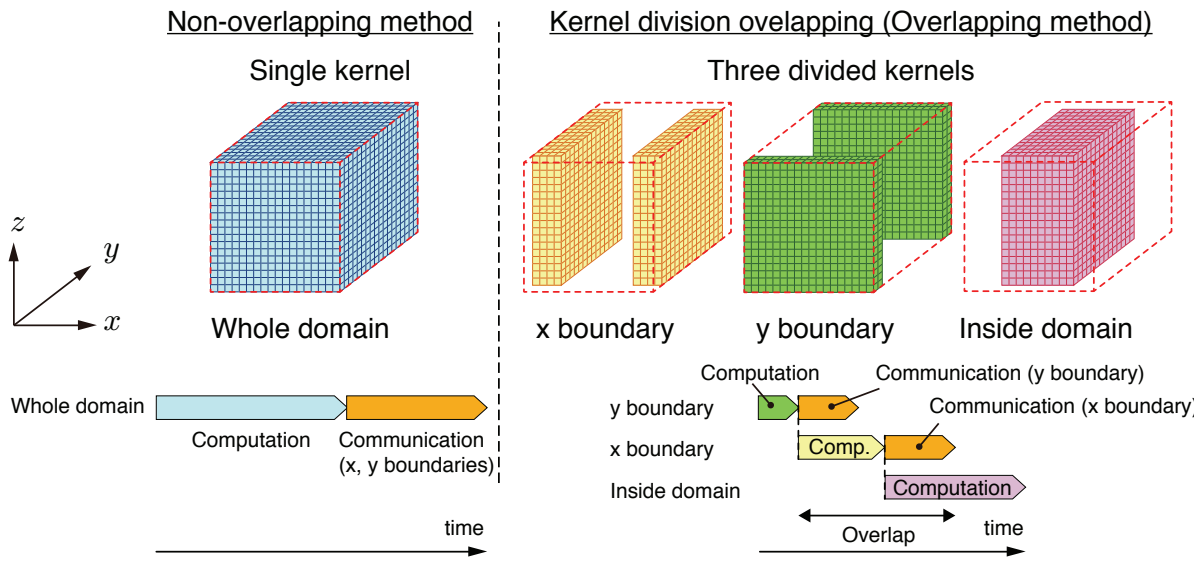

Figure 1: The single kernel for the non-overlapping method and the three divided kernels for the kernel division overlapping in the overlapping method.

5. Performance Analysis and Discussion

5.1. Performance of Single GPU Computing

First, we show the performance of ASUCA using single NVIDIA Tesla M2050 GPU card on TSUBAME 2.0.

In order to measure the performance of floating-point operations on a GPU, we count the number of floating-point operations of the C/C++-based ASUCA by running it on a CPU with a performance counter provided by PAPI (Performance API) [15]. By using the counts and the GPU elapsed time, we evaluate the performance of the GPU computing.

In all the cases, we fixed the grid number $nx = 256$ and $nz = 48$ of the computational domain and varied the number ny from 32 to 208. The performances in both single- and double- precision floating-point calculation were measured on a NVIDIA Tesla M2050 of a node of TSUBAME 2.0. The results of the GPU performance are shown in Figure 2 for the mountain wave test [16] used as a benchmark. In this test, an ideal mountain is placed at the center of the calculation domain. As an initial condition, 10.0 m/sec wind blows in the x direction and normal pressure, temperature, density and the amount of water substances are given. The time integration step is 5.0 sec. All the kernels, including physics processes, used for the real operation except kernels for treatment of real boundary data are executed. Although boundary conditions are updated in real weather forecast, periodic boundary condition are assumed in this mountain test.

We have achieved 49.1 GFlops in single precision for $256 \times 208 \times 48$ mesh on a single GPU. In the double precision, the performance has approximately half the single precision. Each SM in Tesla M2050 has 32 CUDA cores. Each CUDA core has one single-precision unit. For double-precision instructions, two single-precision units are combined so that the peak double-precision throughput is 50 % of the single-precision throughput. Similarly, the double-precision bandwidth would be half of the single-precision one due to the double-element size. From the above, it is reasonable that the double-precision performance has achieved about half the single-precision one in the overall application. As references, the performances on the Intel CPU (Xeon X5670 2.93 GHz 6 cores) are plotted on the same graph. The original Fortran code was compiled by the Intel "ifort" compiler and executed on one socket (6 cores) of Intel Xeon X5670. It is found that the single GPU performance achieved a 12-fold speedups in comparison with one socket of the CPU performance of the Intel Xeon X5670 in double precision.

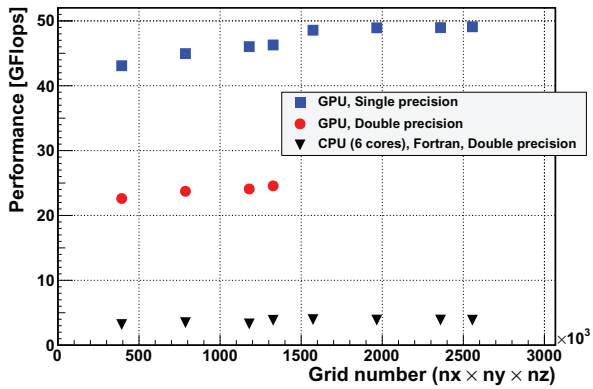

Figure 2: Single GPU performance on a NVIDIA Tesla M2050 in both single- and double- precision. As references, CPU performances of Fortran code executed on one Intel Xeon X5670 socket.

We achieved 44.3 GFlops in single precision using NVIDIA Tesla S1070 on TSUBAME 1.2 with the peak performance and the memory band width of 691.2 GFlops and 102 GByte/s, respectively [11]. Although the theoretical calculation and memory access performances increase by nearly 45 % from a Tesla S1070 GPU to a Tesla M2050, we have observed approximately 10 % of increase in the ASUCA performance from a Tesla S1070 GPU on TSUBAME 1.2 to Tesla M2050 on TSUBAME 2.0. This result suggests that we are able to improve the performance by introducing further optimization specialized for Tesla M2050. Although the peak performance in double precision increases dramatically from 86.4 GFlops of a Tesla S1070 GPU to 515 GFlops of a Tesla M2050, the overall performance of ASUCA in double precision have been improved by about 65 % because ASUCA is a strongly memory bound application.

5.2. Performance of Multi-GPU Computing

We show the performance of ASUCA by multi-GPU computing on TSUBAME 2.0. Bigger domain has better performance and we choose that each GPU handles the domain of 256×108×48 in double precision and 256×208×48 in single precision, respectively. These are the maximum mesh sizes that can be stored on the on-board memory capacity (i.e., 3 GByte) and that minimize uncoalesced memory accesses. The multi-GPU performance of ASUCA is shown in Figure 3. We use three GPUs per each TSUBAME node for these calculations. The numbers of GPUs and the mesh sizes used for multi-GPU computing are shown in Table 1. We measure the performance of ASUCA using both the overlapping method and non-overlapping methods in both single- and double-precision. While computation and communication are performed sequentially in the non-overlapping methods, the three overlapping techniques described above are applied in the overlapping methods to hide communications. We also measure the performance of C/C++-based ASUCA on the CPUs. Note that although this C/C++ code is not optimized enough comparing to the GPU code, we show the performance improved by exploiting the GPUs as a reference. These calculations utilize three CPU cores per each node. Similar to the performance measurements for single-GPU computing, we use the mountain wave test as a benchmark for multi-GPU computing.

As shown in the graph, we achieve an extremely high performance of 145.0 TFlops using 3990 GPUs with the non-overlapping method in single precision. Using the overlapping method in single precision, the performance of 133.5 TFlops is achieved. These results shows that the non-overlapping method has better performance than the overlapping method for multi-GPU computing on TSUBAME 2.0 in single precision, although the overlapping method worked well for multi-GPU computation on TSUBAME 1.2 in single precision to improve the performance of ASUCA [11]. We discuss the reason of this in the next section. Unlike the single-precision calculation, the overlapping method in double precision improves the performance; the overlapping version achieves the performance of 76.1 TFlops on 3936 GPUs, which is higher than 72.1 TFlops with the non-overlapping method. Comparing the single-precision performance with the CPU performance, the performance of 3990 GPUs is compatible with 3990×50 CPU cores.

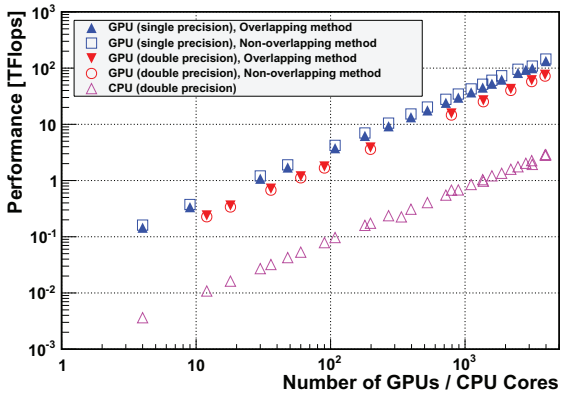

Figure 3: Performance of ASUCA on multi GPUs and CPU cores on TSUBAME 2.0. Note that the reference performance of CPU version is measured by using the C/C++ code.

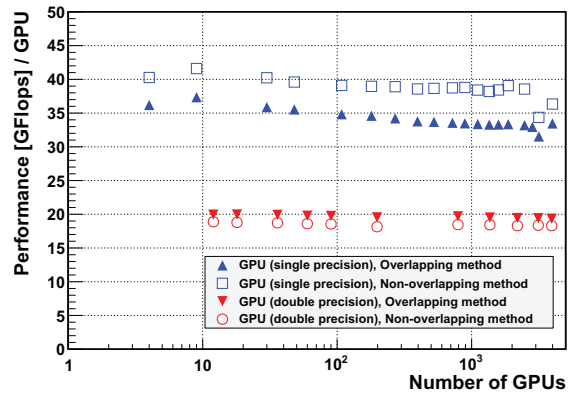

Figure 4: The performance on each GPU when the multi-GPU computations of ASUCA are performed on TSUBAME 2.0.

Table 1: Numbers of GPUs and mesh sizes for multi-GPU computing in single precision (left table) and double precision (right table).

Number of GPUs ($P_x \times P_y$)	Mesh size ($nx \times ny \times nz$)
4 (2 × 2)	508 × 412 × 48
9 (3 × 3)	760 × 616 × 48
30 (5 × 6)	1264 × 1228 × 48
48 (6 × 8)	1516 × 1636 × 48
108 (9 × 12)	2272 × 2452 × 48
180 (12 × 15)	3028 × 3064 × 48
270 (15 × 18)	3784 × 3676 × 48
396 (18 × 22)	4540 × 4492 × 48
525 (21 × 25)	5296 × 5104 × 48
720 (24 × 30)	6052 × 6124 × 48
891 (27 × 33)	6808 × 6736 × 48
1110 (30 × 37)	7564 × 7552 × 48
1353 (33 × 41)	8320 × 8368 × 48
1584 (36 × 44)	9076 × 8980 × 48
1872 (39 × 48)	9832 × 9796 × 48
2142 (42 × 51)	10588 × 10408 × 48
2475 (45 × 55)	11344 × 11224 × 48
2832 (48 × 59)	12100 × 12040 × 48
3000 (50 × 60)	12604 × 12244 × 48
3162 (51 × 62)	12856 × 12652 × 48
3564 (54 × 66)	13612 × 13468 × 48
3990 (57 × 70)	14368 × 14284 × 48

Number of GPUs ($P_x \times P_y$)	Mesh size ($nx \times ny \times nz$)
12 (2 × 6)	508 × 628 × 48
18 (3 × 6)	760 × 628 × 48
36 (4 × 9)	1012 × 940 × 48
60 (5 × 12)	1264 × 1252 × 48
90 (6 × 15)	1516 × 1564 × 48
198 (9 × 22)	2272 × 2292 × 48
336 (12 × 28)	3028 × 2916 × 48
540 (15 × 36)	3784 × 3748 × 48
792 (18 × 44)	4540 × 4580 × 48
1050 (21 × 50)	5296 × 5204 × 48
1368 (24 × 57)	6052 × 5932 × 48
1782 (27 × 66)	6808 × 6868 × 48
2190 (30 × 73)	7564 × 7596 × 48
2640 (33 × 80)	8320 × 8324 × 48
3132 (36 × 87)	9076 × 9052 × 48
3705 (39 × 95)	9832 × 9884 × 48
3936 (41 × 96)	10336 × 9988 × 48

Figure 4 shows the performance on each GPU when the multi-GPU computations of ASUCA are performed on TSUBAME 2.0. The ASUCA code for multi-GPU is confirmed to maintain a good weak scalability. The weak scaling efficiency is above 88 % for 14368 × 14284 × 48 on 3990 GPUs with respect to the 9-GPU performance in single precision. It is found that the basic optimizations for TSUBAME 1.2 are also of benefits to the multi-GPU computing on TSUBAME 2.0.

5.3. Effects of the overlapping method on TSUBAME 2.0

The overlapping method is not effective in improving the performance only in single-precision weak scaling on TSUBAME 2.0 while this method works well for the other cases. As described above, the kernel division overlapping degrades the performance due to decrease of the concurrent threads. We make the difference clear in the breakdown

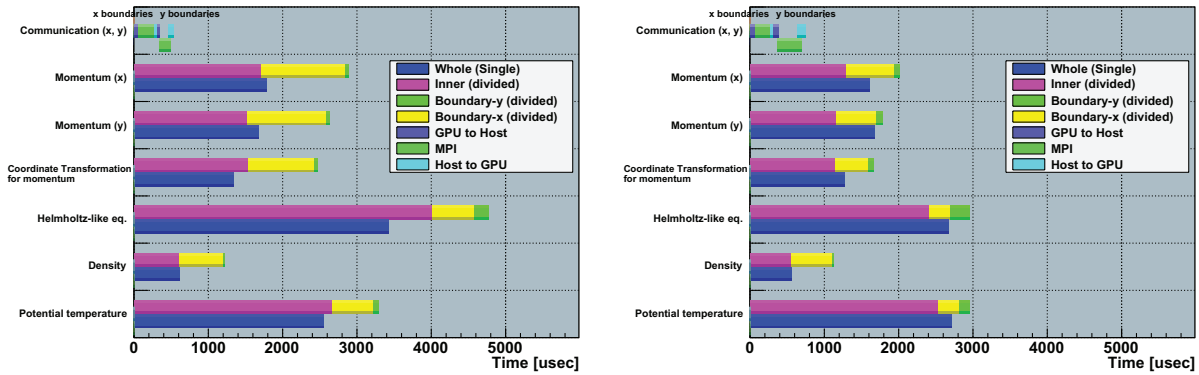

Figure 5: Breakdown of communication times (top bar) and computational times for six groups of kernels in single precision (left graph) and double-precision (right graph). In each of the kernel groups, the upper bar (magenta, yellow and green color bars) displays the computational times of three divided kernels used in the kernel divided overlapping, and the lower bar displays the computational time of an original single kernel for the non-overlapping method. These were measured using 3990 GPUs and 3936 GPUs in single- and double- precision, respectively. A part of the exchange of the y boundaries between CPUs with MPI is performed in parallel with a part of the transfer between a GPU and a CPU.

of the computation and communication times of the kernels adopting the kernel division overlapping among single- and double- precision computations on TSUBAME 2.0 and single-precision computations on TSUBAME 1.2.

Figure 5 shows the communication times and the computational times of six kinds of calculations when 3990 GPUs and 3936 GPUs of TSUBAME 2.0 are used in single- and double-precision calculations, respectively. The execution times of each calculation by three divided kernels where the kernel division overlapping is applied and an original single kernel used in the non-overlapping method are compared. The communication time consists of the elapsed times of the transfer between GPU and CPU, and the MPI communication between CPUs in both x and y boundaries. By using the kernel division overlapping, kernels for the inner region and the x boundary have possibilities to hide the communication time, which is depicted in the top bar of each graph. Since a communication time is shorter than a sum of execution times spent by kernels for inner lesion and the x boundary in all cases, the communication can be hidden with the kernel execution perfectly in both single- and double-precision. Then, the total execution time by performing the three divided kernels is observed as the actual elapsed time in each of the six kinds of calculations. On the other hand, the single-kernel version needs to perform computation first and communication next.

As shown in Figure 5, the kernel division overlapping increases the computational times comparing to the single-kernel versions in all cases due to thread number reduction within each kernel. However, in double precision, the kernel division overlappings are still shorter than the single-kernel versions plus the non-overlapping communication. Thus, the total performance is improved with this kernel division overlapping in double precision in the same way on TSUBAME 1.2. [11].

Unlike these, in the single-precision calculation on TSUBAME 2.0 in the left graph in Figure 5, the increase in the execution time by using the kernel division overlapping from the single-kernel version is longer than the communication time in every case. For example, the amount of this increase in the execution time for calculation of a kernel Momentum (x) is 1102.9 μsec as shown in this graph, whereas the communication, including both MPI and GPU-CPU for the x and y boundaries, spends only 529.8 μsec. Hence, dividing a single kernel into three kernels results in degrading the overall performance, although the communication is successfully hidden by the divided kernels.

The major reason comes from the fact that communication time is significantly reduced on TSUBAME 2.0 comparing to TSUBAME 1.2 while the performance of single GPU has not been improved so much from Tesla S1070 to M2050. Each node is connected via Dual-Rail QDR InfiniBand links (8 GB/s) on TSUBAME 2.0 and there are three NVIDIA Tesla M2050 GPUs attached to PCI-Express Bus 2.0 $\times$16 (8 GB/s), while the nodes were connected via Dual-Rail SDR InfiniBand links (2 GB/s) on TSUBAME 1.2 and there were two Tesla S1070 GPUs accessible through PCI-Express Bus 1.0 $\times$8 (2 GB/s). The communication time of 529.8 μsec on TSUBAME 2.0 is about 19 % of the communication time of 2774.5 μsec observed on TSUBAME 1.2 [11]. Note that this ratio does not directly represent speed-up ratio of communication, because different amounts of data are transferred due to different mesh size

allocated on a GPU. On the other hand, we achieve almost the same performance on both Tesla M2050 of TSUBAME 2.0 and S1070 of TSUBAME 1.2 and, more specifically, 49.1 GFlops and 44.3 GFlops for single GPU computing without communications in single precision, respectively. In our implementation, the minor impact of the overlapping outweighs the benefit in single precision when we use nodes connected to the others via such relatively-fast network as Dual-Rail QDR InfiniBand on TSUBAME 2.0. This observation suggests us that the implementation of kernel division without an extreme drop in the performance is necessary for performance improvements with the overlapping method on TSUBAME 2.0. In the strong scaling, the kernel division overlapping will be effective to improve the performance even in single precision.

6. Real case simulation with ASUCA

This section demonstrates that the GPU version of ASUCA can successfully simulate a basic set of real weather case which is used in the JMA, including the full dynamical core and warm rains. Figure 6 shows the snapshot describing a typhoon with both the real initial and the boundary data used for the current weather forecast at the JMA. This simulation was performed with a $4792 \times 4696 \times 48$ mesh with horizontal mesh resolution of 500 meters over Japan using 437 GPUs of TSUBAME 2.0 in single precision.

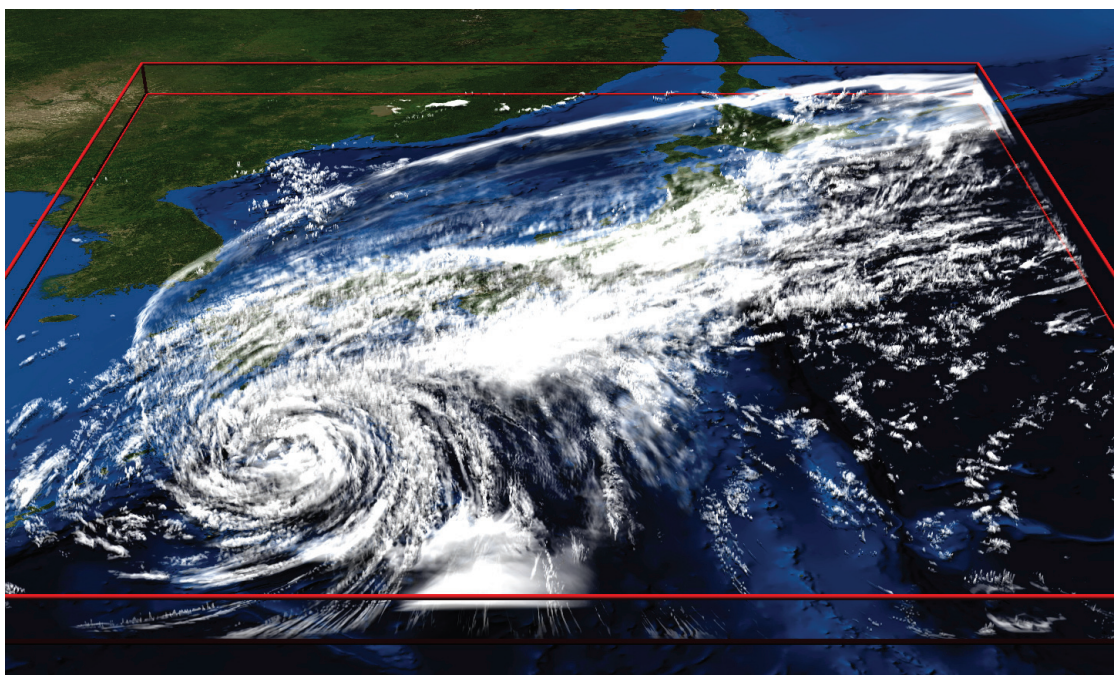

Figure 6: An ASUCA simulation result for a real typhoon over Japan with $4792 \times 4696 \times 48$ mesh using 437 GPUs of TSUBAME 2.0.

7. Conclusion

We have developed the full GPU implementation of the next-generation, production weather code ASUCA and have carried out it on the TSUBAME 2.0 supercomputer in the Tokyo Institute of Technology. The effective utilization of shared memory in the GPU has resulted in the performance of 49.1 GFlops in single precision on NVIDIA Tesla M2050. Our current performance studies exploiting the large number of Tesla M2050 GPUs on TSUBAME 2.0 have successfully demonstrated the weak scalability, reaching the performance of 145.0 TFlops for $14368 \times 14284 \times 48$ in single precision with 3990 GPUs. Although we have confirmed that the basic design of optimization introduced into ASUCA is effective for TSUBAME 2.0, we need to introduce the further optimization specialized for TSUBAME 2.0 into ASUCA in order to make more efficient use of this supercomputer.

Acknowledgment

We would like to thank Prof. Satoshi Matsuoka, Prof. Toshio Endo, Dr. Akira Nukada and Dr. Naoya Maruyama at the Tokyo Institute of Technology for helping us to use TSUBAME 2.0.

This research was supported in part by the Global Center of Excellence Program "Computationism as a Foundation for the Sciences" and KAKENHI, Grant-in-Aid for Scientific Research (B) 19360043 from the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of Japan, and in part by the Japan Science and Technology Agency (JST) Core Research of Evolutional Science and Technology (CREST) research program "ULP-HPC: Ultra Low-Power, High-Performance Computing via Modeling and Optimization of Next Generation HPC Technologies".

References

[1] W. C. Skamarock, J. B. Klemp, J. Dudhia, D. O. Gill, D. M. Barker, M. G. Duda, X.-Y. Huang, W. Wang, J. G. Powers, A Description of the Advanced Research WRF Version 3 (2008).

[2] A. S. Bland, R. A. Kendall, D. B. Kothe, J. H. Rogers, G. M. Shipman, Jaguar: The world's most powerful computer, in: 2009 CUG Meeting, 2009, pp. 1–7.

[3] CUDA Programming Guide 3.2, http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/CUDA_C_Programming_Guide.pdf (2010).

[4] J. C. Thibault, I. Senocak, CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows, in: Proceedings of the 47th AIAA Aerospace Sciences Meeting, no. AIAA 2009-758, 2009.

[5] T. Brandvik, G. Pullan, Acceleration of a 3D Euler Solver using commodity graphics hardware, in: 46th AIAA Aerospace Sciences Meeting, American Institute of Aeronautics and Astronautics, 2008.

[6] A. Nukada, S. Matsuoka, Auto-tuning 3-D FFT library for CUDA GPUs, in: SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, New York, NY, USA, 2009, pp. 1–10. doi:http://doi.acm.org/10.1145/1654059.1654090.

[7] T. Hamada, K. Nitadori, 190 TFlops astrophysical N-body simulation on a cluster of gpus, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, IEEE Computer Society, New Orleans, LA, USA, 2010, pp. 1–9. doi:http://dx.doi.org/10.1109/SC.2010.1.
URL http://dx.doi.org/10.1109/SC.2010.1

[8] J. Michalakes, M. Vachharajani, GPU acceleration of numerical weather prediction., in: IPDPS, IEEE, 2008, pp. 1–7.

[9] J. C. Linford, J. Michalakes, M. Vachharajani, A. Sandu, Multi-core acceleration of chemical kinetics for simulation and prediction, in: SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ACM, New York, NY, USA, 2009, pp. 1–11. doi:http://doi.acm.org/10.1145/1654059.1654067.

[10] J. Ishida, C. Muroi, K. Kawano, Y. Kitamura, Development of a new nonhydrostatic model "ASUCA" at JMA, CAS/JSC WGNE Reserch Activities in Atomospheric and Oceanic Modelling.

[11] T. Shimokawabe, T. Aoki, C. Muroi, J. Ishida, K. Kawano, T. Endo, A. Nukada, N. Maruyama, S. Matsuoka, An 80-fold speedup, 15.0 TFlops full GPU acceleration of non-hydrostatic weather model ASUCA production code, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, IEEE Computer Society, New Orleans, LA, USA, 2010, pp. 1–11. doi:http://dx.doi.org/10.1109/SC.2010.9.
URL http://dx.doi.org/10.1109/SC.2010.9

[12] T. Endo, A. Nukada, S. Matsuoka, N. Maruyama, Linpack evaluation on a supercomputer with heterogeneous accelerators, in: Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10), IEEE, Atlanta, GA, USA, 2010.

[13] K. Saito, T. Fujita, Y. Yamada, J.-i. Ishida, Y. Kumagai, K. Aranami, S. Ohmori, R. Nagasawa, S. Kumagai, C. Muroi, T. Kato, H. Eito, Y. Yamazaki, The operational JMA nonhydrostatic mesoscale model, Monthly Weather Review 134 (2006) 1266–1298.

[14] W. C. Skamarock, J. B. Klemp, Efficiency and accuracy of the Klemp-Wilhelmson Time-Splitting technique, Monthly Weather Review 122 (1994) 2623–+. doi:10.1175/1520-0493(1994)122¡2623:EAAOTK¿2.0.CO;2.

[15] S. Browne, J. Dongarra, N. Garner, G. Ho, P. Mucci, A portable programming interface for performance evaluation on modern processors, Int. J. High Perform. Comput. Appl. 14 (3) (2000) 189–204. doi:http://dx.doi.org/10.1177/109434200001400303.

[16] T. Satomura, T. Iwasaki, K. Saito, C. Muroi, K. Tsuboki, Accuracy of terrain following coordinates over isolated mountain: Steep mountain model intercomparison project (st-MIP), Annuals of the Disaster Prevention Research Institute, Kyoto University 46 B (2003) 337–346.